AFRL-IF-RS-TR-2000-94
Final Technical Report
June 2000

AUTOMATED RESOURCE RECOVERY AGENT (ARRA)

Modus Operandi, Inc.

Michael Winburn

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

AFRL-IF-RS-TR-2000-94 has been reviewed and is approved for publication.

APPROVED: [illegible]

JOSEPH V. GIORDANO
Project Engineer

FOR THE DIRECTOR: [illegible]

WARREN H. DEBANY, Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE JUNE 2000	3. REPORT TYPE AND DATES COVERED Final May 97 - Dec 98

4. TITLE AND SUBTITLE AUTOMATED RESOURCE RECOVERY AGENT (ARRA)	5. FUNDING NUMBERS C - F30602-97-C-0112 PE - 61102F PR - 2301 TA - 01 WU - 02
6. AUTHOR(S) Michael Winburn	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Modus Operandi, Inc. 122 Fourth Avenue Indialantic Florida	8. PERFORMING ORGANIZATION REPORT NUMBER N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFGB 525 Brooks Road Rome NY 13441-4505	10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2000-94
11. SUPPLEMENTARY NOTES Air Force Research Laboratory Project Engineer: Joe Giordano/IFGB/(315) 330-4199	
12a. DISTRIBUTION AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The goal of the Automated Resource Recovery Agent (ARRA) effort is to advance the state-of-the-art in recovery and defense of computer system resources after and during an information attack by developing techniques that bring systems back online quickly. Rather than focusing on plugging the latest holes discovered by hackers, SPS' approach focuses on maintaining system operation through monitoring and recovery of critical resources. While new security breaches will continue to be discovered, focusing on the recovery and defense of the targets of these attacks increases the chance of system survivability, shortens recovery time, and minimizes the impact of newly discovered methods of attack. In addition to automated resource recovery, our concept is to assume a defensive posture upon detection of malicious activity to help safeguard the system. Using heuristic methods to analyze system resources, we will focus not just on individual resources in isolation, but reason about the interrelationship of the various components and system operational concepts. By using generalized heuristics, an effective course of recovery and defensive action can be developed that addresses the source of the problem and not just the symptom.

14. SUBJECT TERMS Intelligent Agents, Automated Recovery, Information Warfare, Computer Security, Resource Monitoring			15. NUMBER OF PAGES 44 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

Table of Contents

List of Figures

ARRA: Project Overview

The goal of the Automated Resource Recovery Agent (ARRA) effort was to advance the state-of-the-art in the recovery and defense of computer system resources after/during an information attack by developing techniques to quickly bring systems back online. Rather than focusing on plugging the latest holes discovered by hackers, our approach focused on maintaining system operation by monitoring and recovering critical resources. While new security breaches will continue to be discovered, focusing on the recovery and defense of the targets of these attacks increases the chance of system survivability, shortens recovery time, and minimizes the impact of newly discovered methods of attack. In addition to automated resource recovery, the ARRA agent can assume a defensive posture upon the detection of malicious activity to help safeguard the system. Using heuristic methods to analyze system resources, the ARRA agent not only monitors individual resources, but reasons about the interrelationships of the various components and system operational concepts.

The ARRA project brought together expertise in distributed computing, computer security, C4I systems, and artificial intelligence to provide a new approach to the automated recovery and defense of critical system resources. The commercial standard for recovery of disabled or corrupted critical resources is to maintain backup systems. In terms of resource duplication, this method of recovery is expensive. Further, it doesn't always recover resources in a timely manner.

Our project resulted in the design, development, and demonstration of an extensible agent architecture that implements the recovery concepts of *warm-start* and *hot-start*. The application of this technology was demonstrated during the final Recovery IPT meeting in November 1998. During the demonstration, it was shown how the ARRA is configured, how it monitors resources, and how unattended automated recovery is accomplished using agent technology.

Objectives

The specific objectives of the ARRA project included: (1) investigating and defining specific critical resources that are susceptible and vulnerable to information warfare attack, (2) designing and developing methods for monitoring critical resources, (3) developing heuristic methods, using AI technology, for ensuring that critical resources are available and operating correctly, and (4) developing a prototype implementation to test, validate, and demonstrate our concept.

Background: Intelligent Agents

The term *agent* or *intelligent agent* is used to define software capable of acting in a specified way to accomplish a user-defined task. To differentiate intelligent agents from *other* fields in Artificial Intelligence or distributed processing, academia has attempted to define key characteristics of agents. Intelligent agents can be classified by their mobility (either static or mobile), their level of deliberative capabilities (ranging from an internal reasoning model to having the ability to coordinate with others), and their reactive capabilities. Intelligent agents also possess several characteristics that help differentiate them from distributed processing software. *Autonomy*, a characteristic commonly associated with agents, is the ability to operate without the need for human guidance. This allows an agent to behave in a proactive manner. Another key characteristic is the ability to *learn*. This exists at some level in all intelligent agents, and is another important discriminator when comparing agents to distributed processing. Learning is accomplished by interacting with the environment or with other intelligent agents. These interactions with other agents demonstrate the last key characteristic, *cooperation*. Agents tend to cooperate among themselves via communication of a current task or a condition discovered. This type of communication generally occurs at a high level, unlike the low-level protocols used by distributed processing. Along with the key characteristics described above, several minor characteristics can be used to categorize intelligent agents. These include *discourse*, usually implemented using a formalized set of interactions that define the tasks to be accomplished, *graceful degradation, domain-specific knowledge*, and the software equivalent of *trust*.

The ARRA agent embodies many of these characteristics. Each agent operates autonomously without the need for human interaction as it collects information about its environment. The analysis expert system uses generalized heuristics and models of nominal system behavior when analyzing and reasoning about the current state of the operational environment. Corrective courses of action are developed through environmental analysis. The agents carry out these actions to perform resource recovery, resource defense, or to modify future behavior by updating the agenda of tasks to perform.

Each ARRA agent has the ability to coordinate with other agents in the domain. For example, as part of the resource protection and recovery process, agents coordinate the dispersal of critical files among themselves using a "request for volunteers" metaphor. When an agent receives a file-monitoring request, it broadcasts a message requesting volunteers to provide backup for its new file. Other agents in the society "volunteer" to help out, and the file is dispersed to a select number of agents based on the priority and criticality of the file.

Information about the state of individual agent environments is also shared among the agents in the society. When a security breach is discovered, an alert is sent out to the other agents. This information enables the other agents to "be on the lookout" for other similar breaches in their environments. These interactions among the agents demonstrate the key characteristics of agent behavior and distinguish them from other types of software.

ARRA: Technical Approach

ARRA monitors, on a per node basis, the resources necessary to carryout the desired functioning of a computer system. These resources include system processes, mission critical applications, configuration files, binary executable files, dynamic files, file system integrity, and other system resources. Figure 1 illustrates this concept.

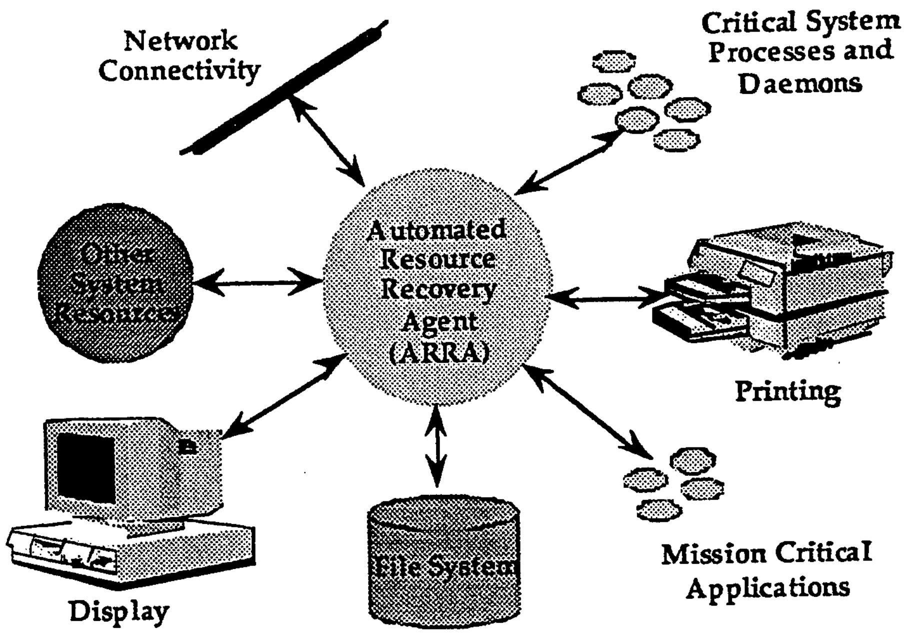

Network Connectivity
Critical System Processes and Daemons
Automated Resource Recovery Agent (ARRA)
Other System Resources
Printing
Display
File System
Mission Critical Applications

Figure 1. Automated Recovery of Critical Resources Concept

The Automated Resource Recovery Agent is shown at the center of the diagram. It iteratively gathers the appropriate indicator and status values for critical system resources. The agent then uses rule-based heuristic methods to analyze and determine the current state of the system. If the agent discovers that a critical resource has been compromised, it automatically takes an effective course of action to reinstate the system to its operational configuration. The agent also assumes a defensive posture to guard against repeat attacks on the affected resource. Having recognized the possibility that an attack may be underway, the agent operates at a higher level of alert, increasing its surveillance of critical resources and being more *suspicious* of system activity.

Agent Architecture: A Functional View

This section presents a detailed view of the functioning and interaction of the components that form the foundation of the agent architecture. We describe how the agent uses declarative and state information to determine whether an attack on the system has occurred or is occurring. We then detail the mechanisms used to initiate recovery and reinstate the system to its operational condition. We also describe the use of *camouflage* to protect vital system and mission critical files. In addition, we discuss the use of defensive action to limit or eliminate newly discovered threats to the system, as well as how the ARRA agent uses heuristic methods to implement concepts such as threat levels, temporal reasoning about events, and the general nature of system operation. Each component part of the agent is discussed, along with its relationship to the other components. Taken together, this provides a detailed understanding of our overall concept for the automated recovery and defense of critical system resources.

To meet our objectives, we designed and developed a generic agent architecture as shown in figure 2. This architecture consists of the following elements:

- A Resource Monitor that iteratively gathers state information about the system being monitored
- An Analysis Expert System that uses heuristic methods to reason about the system state
- An Action Control component to carry out actions that have been deemed by heuristics as necessary for the well being of the system
- An External Agent Interface to provide a communication path for the agent to send and receive messages
- A database to contain information about the current system state and trend, and a system model to describe the nominal state of the system
- A dynamic Agenda that describes the work to be performed by the agent

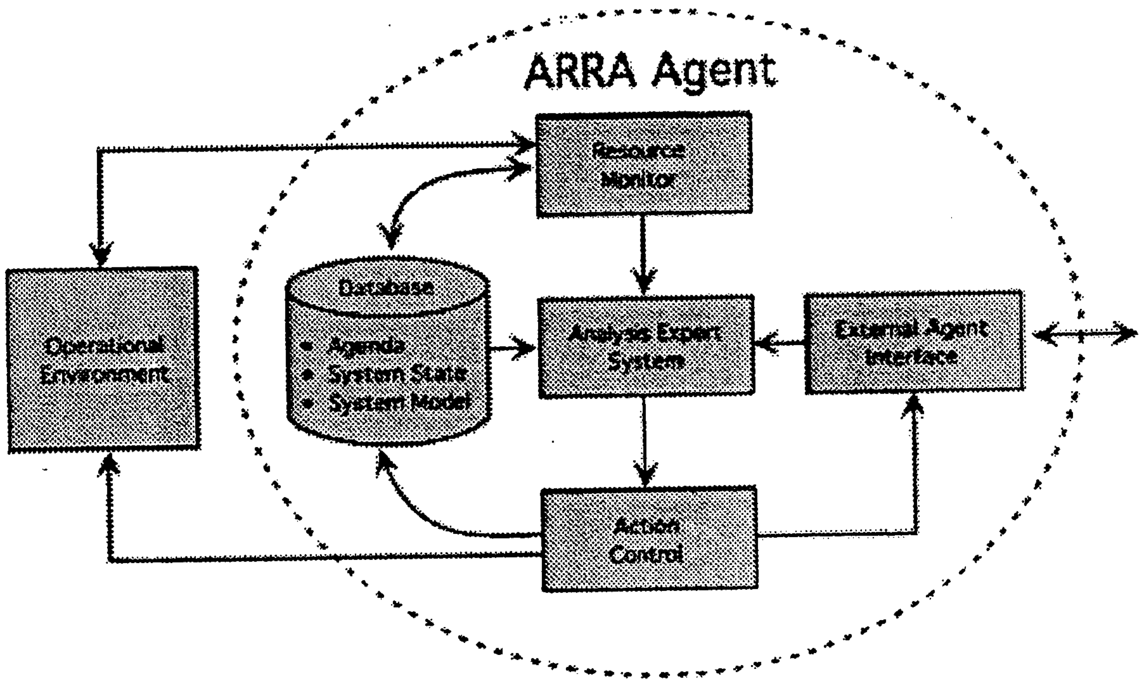


Figure 2. Agent Architecture

Resource Monitor

The primary function of the agent's **Resource Monitoring** component, shown in figure 3, is to iteratively gather information about the state and functioning of an operational computing system.

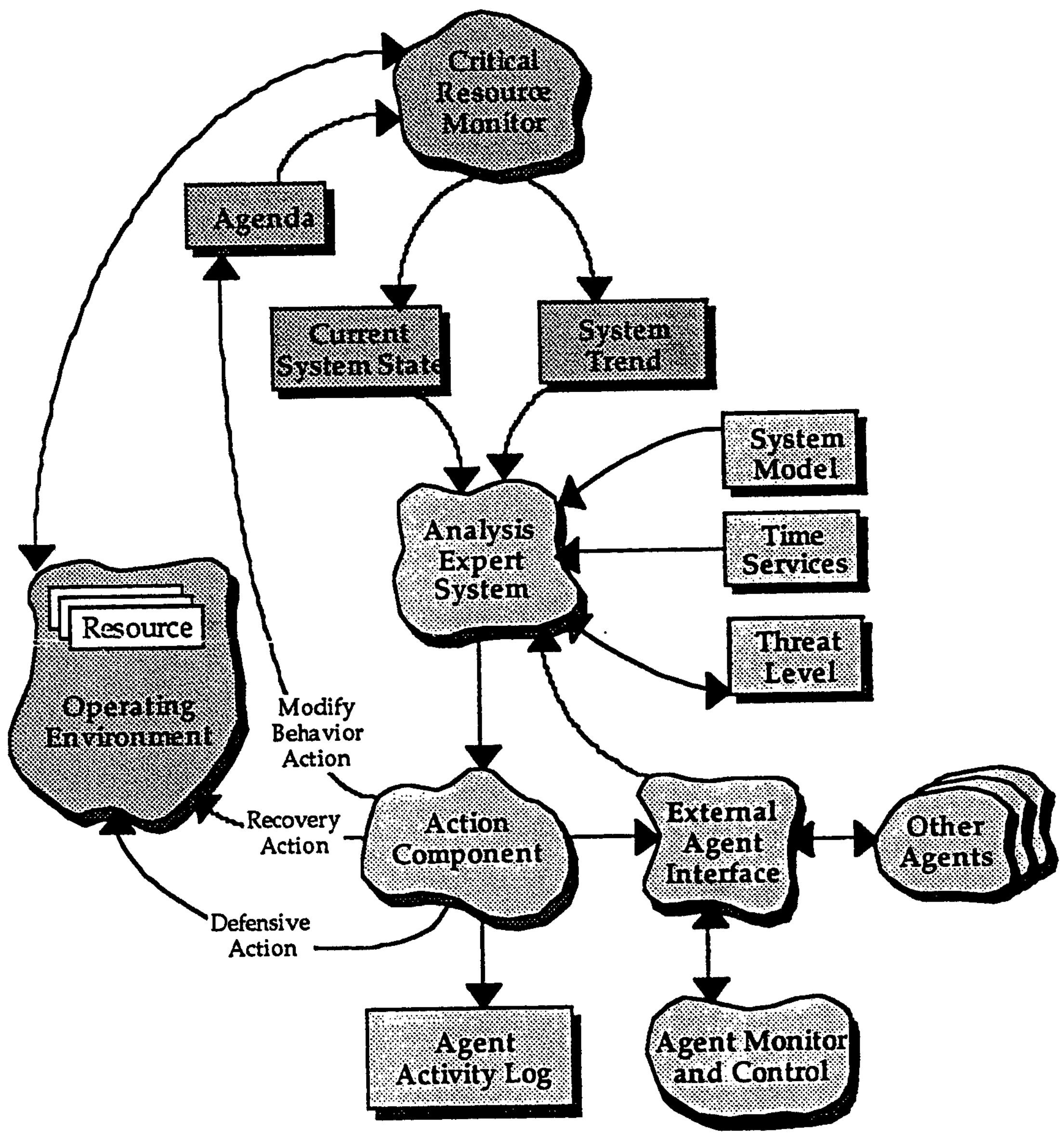


Figure 3. Resource Monitor

The agent performs this activity using the facilities of the operating system to retrieve information about processes and files. It collects resource information based on commands given to the Agenda, such as the name of the resource and how often to gather status. Once the information is gathered, it is stored in the Current System State table. The Resource Monitor does not perform any evaluation of the data that it collects. That is performed by the Analysis Expert System, which is discussed in the following section.

Three data areas are associated with the Resource Monitor: 1) the Agenda, 2) the Current System State, and 3) the System Trend.

The Agenda, Current State, and Trend

The **Agenda** provides a method of specifying which resources are to be monitored and how often to gather status for each resource. The Agenda is an *object* in the true object-oriented sense. Rather than being a static table of values that are read and acted on, the Agenda – upon receiving a message to monitor a new resource – creates a new instance of the requested resource monitoring class as an independent thread process. The new instance has all the active components of the generic agent architecture: Resource Monitor, Analysis Expert System, and Action Component. Each newly created agent operates on a single resource to monitor, analyze, recover and defend from attacks. This mechanism allows each agent to independently adjust its actions and behavior dynamically as external conditions warrant. Each agent shares information about its resource state through the commonly shared Current State Table. Overall system analysis is performed by the overseer AES and is described later in this report. The **Current State** table provides an area to store the most recently collected resource values, and serves as short-term memory for the agent. The **System Trend** table contains historical and computed values that maintain a more "long-term" view of the state of the resources over time. By maintaining both short-term and long-term memory, the agent can reason about the system state not just from a single snapshot in time, but also from an historical perspective.

Analysis Expert System

The **Analysis Expert System** appears in the center of figure 4. This component provides intelligence and reasoning capabilities to the agent. Implemented using rule-based expert system techniques, generalized heuristics guide the analysis and evaluation of the overall system state to ensure that critical system and application resources are available and operating properly. By developing and using generalized heuristics, our approach focused on operational concepts, *not* the implementation of single purpose, single action if-then rules. New relationships can be discovered by employing heuristics both to suggest plausible actions and to prune implausible ones. To accomplish this, requires heuristics of varying levels of generality and power, an adequate representation for knowledge, some initial hypotheses about the nature of the domain, and the ability to gather data and test conjectures about the domain.

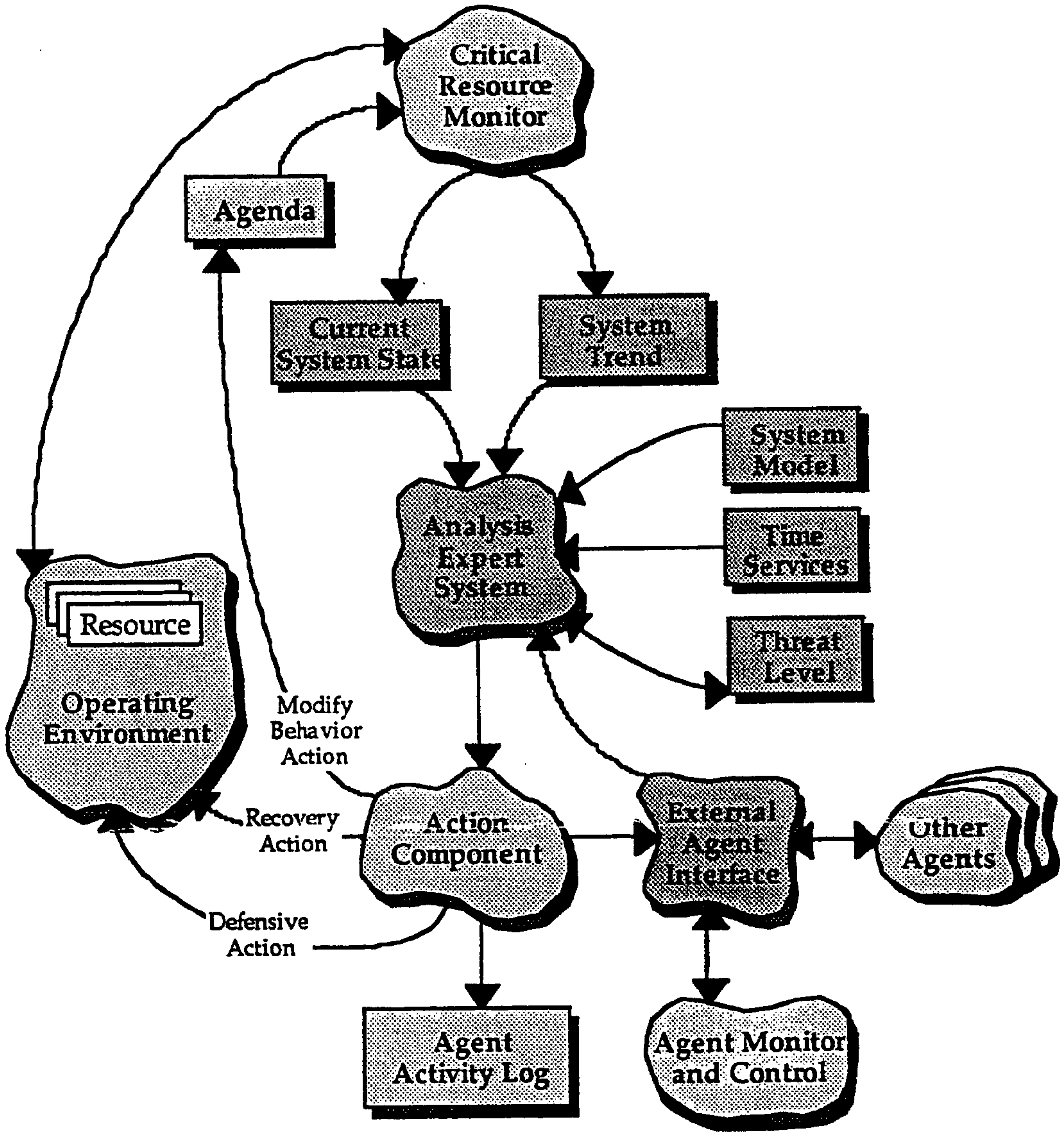


Figure 4. Analysis Expert System

To carry out intelligent and timely automated recovery, the agent must understand the concept of *system operations*. Our approach used heuristic methods to develop a "big picture" of the interrelationship of the various component resources. By reasoning about the system as a whole, and not just individual resources in isolation, an effective course of recovery and defensive actions were developed.

The Analysis Expert System (AES) component uses information from six sources to perform intelligent system analysis: 1) Current System State table, 2) System Trend table, 3) System Model, 4) Time Services, 5) Current Threat Level, and 6) External Agent Interface.

The primary sources of system state information are the Current System State and Trend tables, which we discussed in the previous section. These tables contain status information collected by each Resource Monitor. The AES heuristics use this information in concert with the System Model to determine system condition and the need for automated recovery and defensive action. The System Model contains parameter values that describe the operational state and interrelationship of the resources being monitored. Changing parameters in the System Model provides a way to adjust agent behavior without having to modify the rule base.

Time Services

Our approach implements the concept of time-related monitoring and analysis using **Time Services**. Computing environment activity varies throughout the day and from weekdays to weekend. By using a time-based approach, monitoring activity can be scheduled for the time of day and day-of-week to provide more oversight and restricted access during times when the computer system is unattended. For example, changes to the password file can be eliminated during off-hours to help safeguard the system from attack.

Modeled after the UNIX "cron" facility, the "CronEntry" object is used to give an added dimension to the standard Agenda Item. ARRA's modeling of the UNIX cron facility uses a single CronD object, which models the crond daemon, and a collection of CronEntry objects. CronEntry objects are:

- A five-component time specifier: minute, hour, day of week, day of month, month
- A one-shot flag to indicate the action should be executed only once
- The action to be executed
- The AgendaItem on which to operate

The CronD is simply a list of CronEntry objects. At any time, entries can be added or removed from the CronTable via the user interface. The CronD executes as a thread, examining all CronEntry objects once every 60 seconds (as does the UNIX crond), and executing the actions for any active CronEntry objects.

The CronD does not understand time specifiers, and does not examine each CronEntry in any detail. Rather, the CronD simply asks the CronEntry whether it should currently be active. The CronEntry then internally checks each of its fields against the current time, and indicates that it should be active if all fields match. If the CronEntry is active, the CronD then instructs the CronD to execute its specified command.

Threat Level

The fifth source of input to the AES is the **Threat Level**. Employing threat level concepts, the AES can dynamically modify its behavior and analysis based on identified threats. This can be as a result of executing a recovery action or from detecting suspicious activity through analysis of the system state. The AES modifies (increases/decreases) the threat level based on the execution of rules. For example, when an agent detects that a configuration file has been compromised, it increases the threat level. Other agents, detecting this change, can increase the frequency at which they monitor their respective resources to limit the damage and promptly respond to future attacks. After an appropriate (a pre-specified) amount of time has passed without further attacks being detected, the Threat Level is lowered and normal monitoring frequencies are reestablished.

External Agent Interface

The last input to the AES is from the **External Agent Interface**. This provides a way for the agent to send and receive information external to the agent, such as from other agents and from an agent operator. Agents use the EAI to communicate actions, such as a change in threat level or a detected attack, among agents operating on different computer platforms within the network. This kind of communication can be used to signal the agent society to "be on the lookout" *before* they are attacked.

It is also desirable to employ a degree of external control and monitoring of agent activity by a system operator. An authorized operator can give resource-monitoring requests directly to the agent through the user interface.

The AES uses these six sources of input data to determine the operational state of the system and to make decisions as to what actions are necessary to maintain operational conditions. The results of a particular thread of analysis by the AES can have four conclusions:

1) No action is necessary -- All systems are operating normally.

2) Modify agent behavior -- This conclusion is generally the result of a suspected malfunction or attack, a threat level change, a time mode change, etc.

3) Take defensive action -- This conclusion is generally the result of a suspected attack, and causes action that modifies the computing environment, such as disabling external logins.

4) Take recovery action -- This conclusion is generally the result of a determination that the computing system is not operating in its desired state, and causes corrective actions to take place.

After the AES has reached a conclusion based on analysis of the input, data instructions are sent to the Action Component to carry out the request.

Action Component

The agent's **Action Component** (AC) is shown in the lower portion of figure 5. The purpose of the Action Component is to carry out actions that the AES deems necessary for the well being of the system being monitored. When a change in activity is required, the Action Component receives instructions from the Analysis Expert System based on evaluation of the current system state. The action categories correspond to the last three of the four conclusions that can be reached by the AES. Since no action is required as a result of the AES' conclusion that all systems are operating normally, no corresponding activity is required by the AC.

The following three categories are supported:

1) Actions that modify agent behavior
2) Actions that put the system in a defensive posture
3) Recovery actions that restore the system to its operational state.

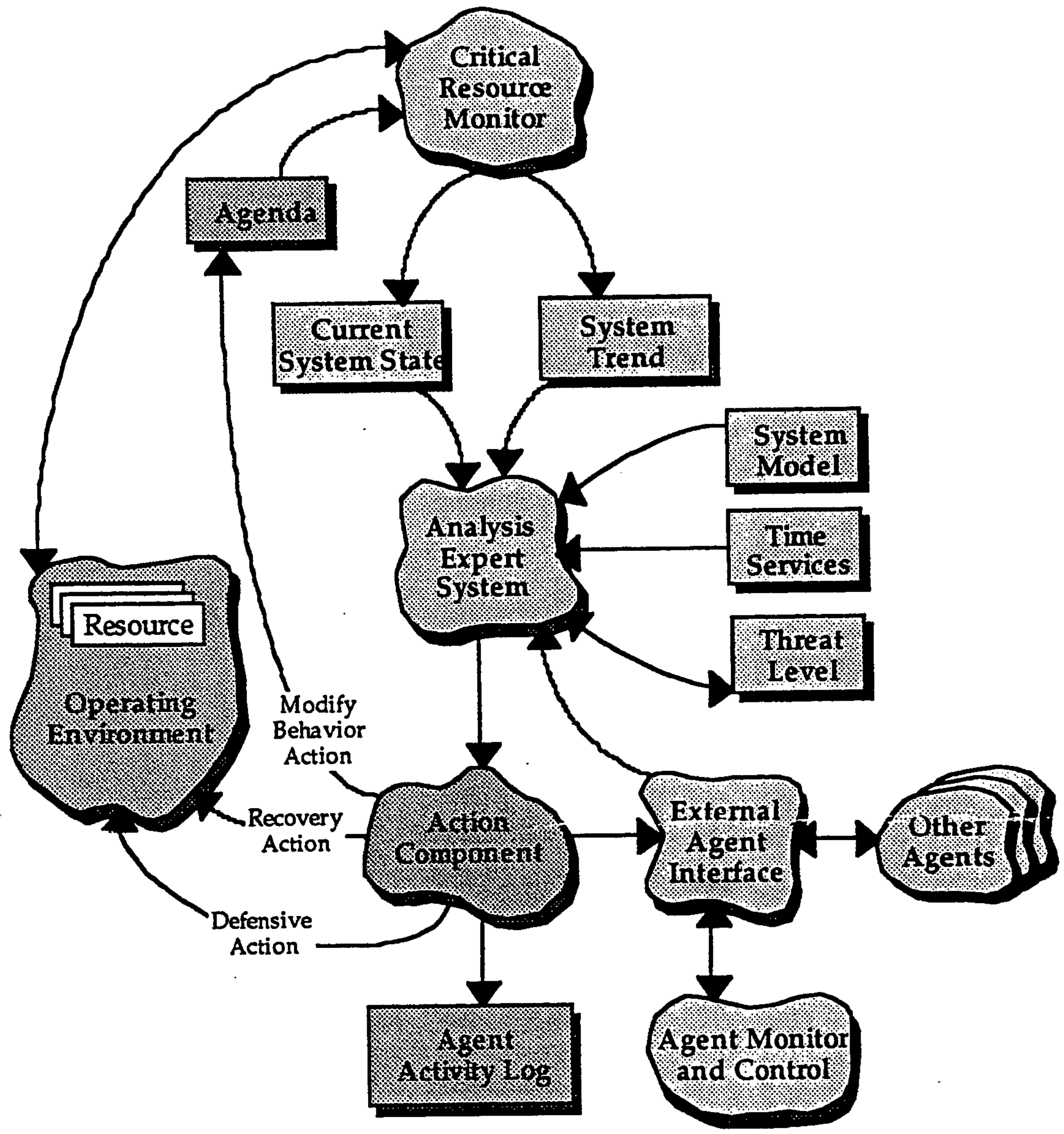


Figure 5. Action Component

Agent Behavior

The Action Controller, through changes to the Agenda, modifies agent behavior. The Agenda, by instantiating single-purpose agent threads, controls what and how often status is gathered about the system resources. By adding or deleting items from the Agenda, the types of status information can be dynamically controlled. In addition, changing the sampling rate at which the status is collected provides a means of focusing/defocusing attention on a resource. This is illustrated in figure 5 by the line between the Action Component and the Agenda.

Recovery

When the agent discovers that a critical resource is missing or has been altered, recovery actions to restore the system to its normal operational state are initiated. When it is determined that a monitored application is no longer present in the system – whether from program failure or maliciously intent – actions are taken to perform initialization, restart the critical application, and restore the operating environment.

Camouflage

Configuration data and binary executable files are monitored to ensure they have not been altered. If it is determined that modifications have been made to these files, file recovery actions are performed. This is accomplished using a file camouflage technique. To restore a file, such as a critical application's binary executable, a copy of the file must be available for restoration. If an intruder has tampered with a file, special caution must be exercised to ensure that the backup copy has not been altered as well. To make it more difficult for the intruder to find the backup file, it is camouflaged by:

- Encrypting the file to protect it from being found using binary file comparison techniques
- Compressing the file to minimize storage requirements and to change its size
- Naming the file something other that its target name
- Storing the file in a directory other than its target directory
- Computing a message digest value for the file to be used to ensure that the file has not been altered

The use of this technique minimizes the possibility of an intruder corrupting both the original and the backup file. While it is not desirable to duplicate every file in the system, the camouflage technique provides a way to ensure the integrity of selected files, and provides an automated method for the recovery and protection of key system files and process binaries.

All actions to the operating environment are centralized in the Action Component. This limits the point (within the architecture) at which the operational system can be modified.

Each action taken by the agent is written to the Event Log and can be viewed using a word processor or through the user interface. The agent log file is monitored as critical files to ensure that it is not modified by an intruder.

Resources

ARRA focuses on two main categories of system resources that are monitored by the agent – **processes and files.** Based on our experience in distributed computing, network management, and system control, we identified a set of candidate resources in each of the general categories. In the following paragraphs, we discuss the purpose of each category selection and list the supporting resources.

The purpose of monitoring processes is to ensure that mission critical applications and system processes are running and operating as desired. While ARRA can determine that these processes are running by gathering and evaluating process statistics, it has little control of the internal functioning of an application. The agent can, however, monitor how a process uses critical, finite system resources to determine if there is a "run-away" process or a resource-consuming virus.

Information about process state is gathered using operating system APIs. This includes information such as the process owner, files that are currently open and being accessed by the process, CPU utilization, and system resources that are being used by the process. Some information about processes is not accessible through the operating system. A good example is declarative information – such as a process type – that indicates whether a process is a system processes, a mission critical application, or a user process. This type of information is maintained in the System Model, which is the repository for declarative system information. Both declarative and statistical values are necessary for the evaluation of the health and welfare of the system being monitored.

Files are monitored to ensure that binary executables, mission and operating system data, and configuration files have not been altered or corrupted. It is necessary to monitor these files to guard against tampering or substitution of a system executable with another file having the same name, but with different – possibly malicious – execution results. There are many forms of malicious software: Trojan horses, viruses, worms, trap doors, time bombs, logic bombs, and others. While there are semantic differences among the forms, the result is usually the same – system compromise. The Trojan horse's modus operandi is that it appears to be an ordinary program that performs a useful function while also surreptitiously performing malicious activity. Our approach is to protect against the introduction of these forms of system compromise by ensuring that only authorized files are present in the system. This is accomplished using a combination of file monitoring and file camouflage, as explained in the Action Component section, which describes file recovery.

Agent Design and Development

The agent architecture was designed using an object-oriented methodology. This was done for several reasons. One goal of our design was to develop an architecture that was generic and could be easily extended to monitor and recover other resource types. In addition, the generic nature of the design allows agent technology to be applied and reused for purposes other than automated recovery of resources. After researching existing approaches and experimenting with various designs, we decided that the most flexible architecture would be one comprising small, single-purpose agents that separated the work to be performed by the agent from the underlying infrastructure components. The result was an architecture that had the core agent components replicated for each resource to be monitored/protected/recovered. Each of these single-resource agents in turn is monitored by an overseer agent that analyzes the state of each of the sub-agents to arrive at the state of the entire system. This is shown in figure 6.

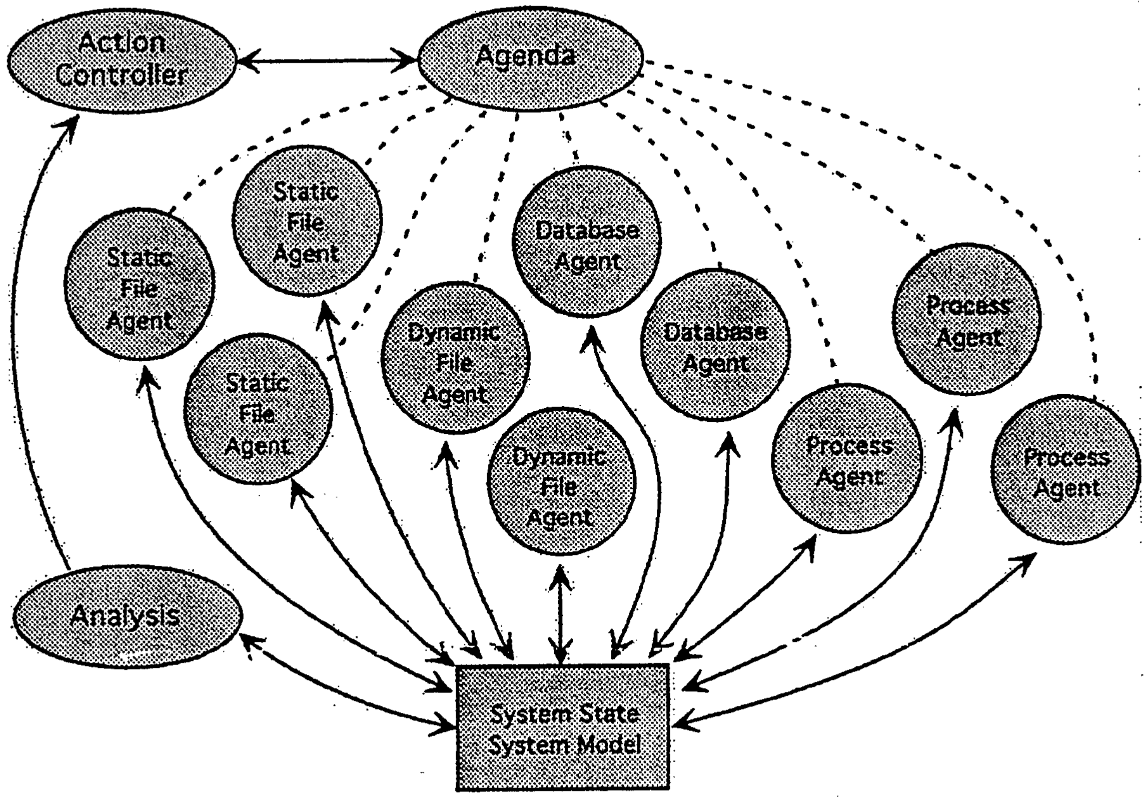


Figure 6. Multiple Single-Purpose Agent Architecture

Figure 6 shows agents for each static file, dynamic file, process, and database – with the overseer agent indicated by the Agenda, Analysis, and Action Controller. The sub-agents communicate to the overseer through the System State table, as shown at the bottom of the figure. This architecture meets our goal of extensibility. To add new monitoring capabilities to the agent – for example, monitoring network traffic – the only requirement would be creating a network monitor class and inheriting the existing core functionality of the agent.

An implementation of the architecture just described can protect, monitor, and recover resources operating on a single computer platform. The addition of the External Interface allows agents to extend their capability to communicate and operate in a distributed environment. This is a key feature of the extensibility of the architecture. Through the External Interface, shown in figure 7, agents operating on computing platforms within a local area network can communicate and exchange information about the state of their individual environments. Information such as "critical processes are being compromised on computer 123" can be broadcast to other agents within the society. These types of alerts provide information that can have a global impact on the protection of the entire computing environment. Agent behaviors are implemented using the rule-based capabilities of the Java Expert System Shell that is integrated into the Analysis component . Individual agent and agent society behaviors can be created and tailored by creating new rule sets. This provides a flexible mechanism to extend the capabilities of the agents without requiring any modification to the underlying core implementation.

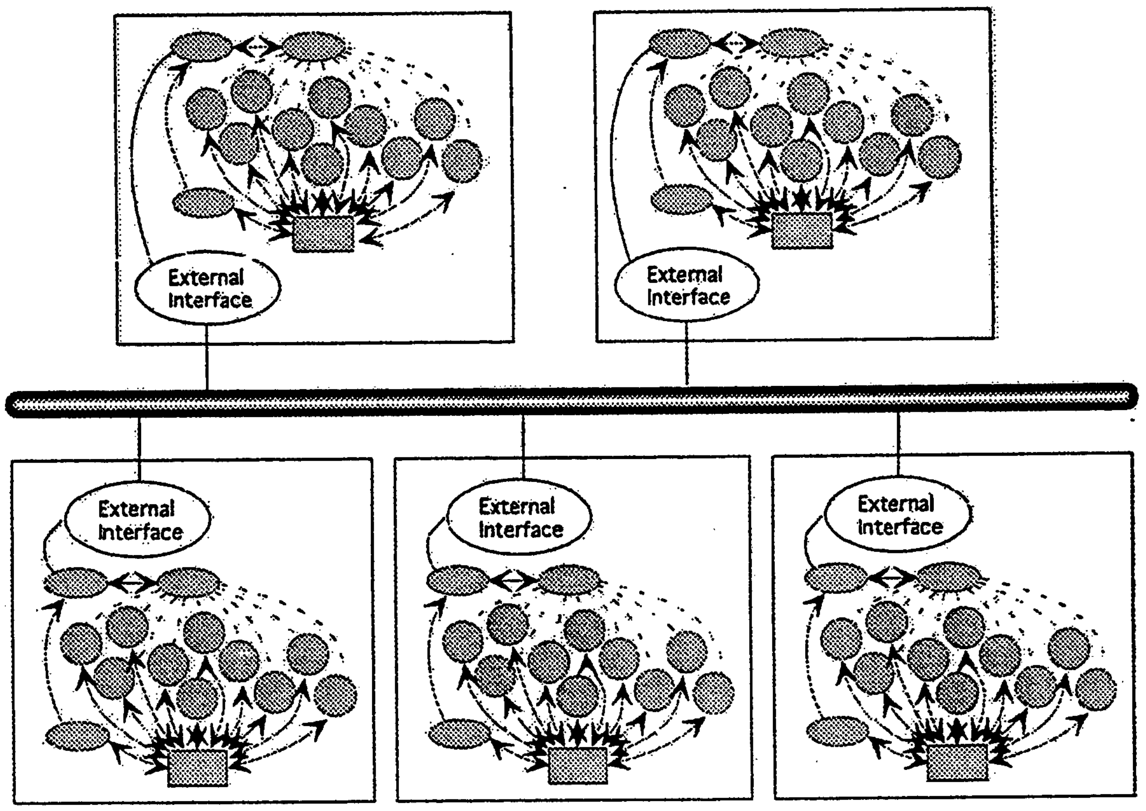


Figure 7. External Interface

Implementation Language

Implementation of the architectural design was done using the Java programming language. Java was selected because of its ability to execute on a variety of platforms without requiring modification or recompilation. Java provided a natural fit for the object-oriented design used to create the agent architecture. The language also provided predefined classes that were used to implement agent communication via the User Datagram Protocol.

Additionally, using Java to implement the User Interface provided portability between platforms, which would have been difficult using standard GUI environments, such as Motif or native MS Window APIs.

The User Interface and Dialog Management

During initial development, a very flexible and open form of User Interface (UI) management was developed. The individual UI components use the document-view pattern, a variant of model-view-controller (MVC). The ARRA UI package contains an interface that is a specification for creating "Viewable" objects. Any Viewable object can be represented, and optionally manipulated, with self-defined UI components. These Viewable objects are managed with a central ViewableDialog component. Each Viewable component can be represented with either a button or a menu item, and when selected will be displayed in a unique dialog window.

This design and implementation approach meant that any single ARRA component – no matter how big or how small – could define itself as viewable and could be readily integrated with the rest of the ARRA UI – without modification to the other UI components. The decoupled nature of the design (i.e., very little dependence of one UI component on another), permitted the quick addition of new components and the quick reconfiguration of the general layout of the UI. This made preliminary UI development very easy. Further, changes to the underlying data model of the ARRA, such as the addition of the AnalysisExpertSystem component, did not require any changes to the UI.

As the ARRA matured and development began to stabilize, the need for such a disconnected UI did not weigh as heavily. Also, the relative inconvenience of having a separate dialog for every individual UI component strongly indicated the need for a "cleaner" approach to UI organization. While some parts of the UI – specifically, the short-lived dialogs for entering specific information, such as creating a new CronEntry – still needed independent dialogs, most UI components did not. We determined that a single shared dialog would be sufficient for most UI components.

The "new" UI management structure is not more restrictive than the "old" structure; in fact, the internal message structures are virtually identical. Most changes involved moving entry-point functionality from the ViewableDialog to the ArraMenuBar.

The new shared dialog still consists of many independent views of the components, but it uses a set of tabbed panes – as shown in figure 8 – rather than independent lightweight dialogs.

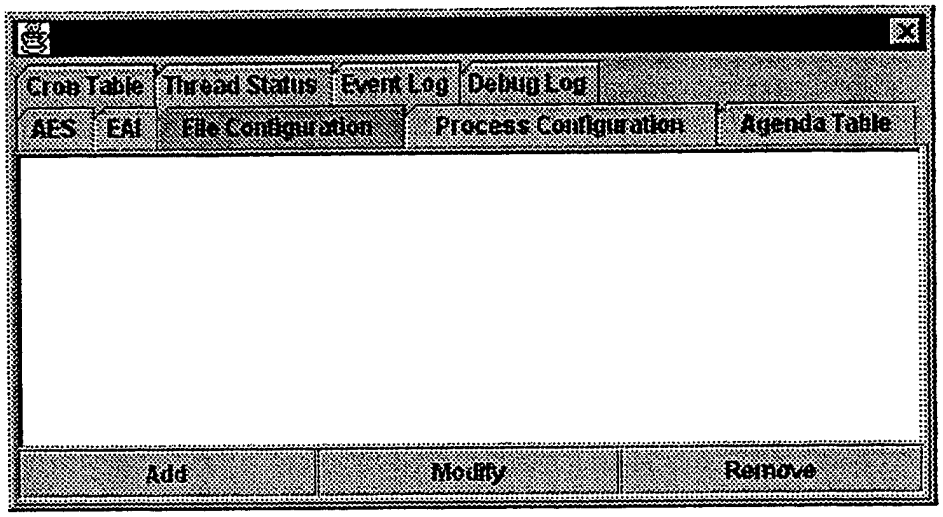


Figure 8. Tabbed UI dialog

The placement of this shared set of panes resides in a single dialog. As with the older style, the shared dialog is displayed when an item is selected or selected from the menu bar. After a selection is made, the corresponding tab in the shared pane is brought to the front. The dialog can be hidden at any time without impacting the functionality of the agent.

Analysis: The Java Expert System Shell (JESS)

The Analysis Expert System (AES) is an independent component of the ARRA. It makes "intelligent" decisions about the actions and activities of the ARRA. The AES consists of several smaller sub-components: the Java Expert System Shell and its control script, arra.clp; the AES itself, which is the interface between the ARRA and JESS; and the Trend, which serves as a sort of "memory" for the AES.

JESS uses a command language that is a subset of that used by the CLIPS expert system shell. CLIPS was developed a number of years ago by NASA. The language syntax is similar to LISP but the original implementation was done in the C programming language. JESS, developed at the Sandia National Laboratory, is a Java port of CLIPS.

JESS can be programmed using Java or in the CLIPS command language. In our case, we used a combination of the two. The AES adds a few functions to help interface with JESS, including methods that add state information to JESS and methods that give JESS the ability to interact with the ARRA. The arra.clp file contains most of the logical operations used in analysis by the ARRA. The Trend tracks the histories of all the active AgentThreads in the ARRA and provides a few high-level operations for manipulating the actions and behaviors of the ARRA.

The Trend is relatively simple, considering the complex task it performs. It consists of a list of AgentThreadHistories – one for each AgentThread in the ARRA. Each AgentThreadHistory is itself a queue of the past states of an AgentThread. The Trend supports several operations called from JESS, namely an "increase-freq[uency]" and a "decrease-freq[uency]". The Trend does not blindly increase or decrease the polling frequencies of AgentThreads upon direction from the JESS, however. The direction of the JESS is just that – direction – and is used by the Trend as a *suggestion* of how to handle the AgentThreads.

Upon receiving the "increase-freq" message about a particular AgentThread, the Trend will only increase the frequency (or, more precisely, decrease the period) to once per 5 seconds (periods will not fall below 5s). Currently, the Trend simply divides the period by 2 and sets the new period to the lesser of the quotient and 5. In future development, the Trend should also consider a "sensitivity" for each AgentThread, where a higher sensitivity would necessitate more aggressive changes to the polling frequency. This "sensitivity factor" could be dynamic and based, at least partially, on the history of the AgentThread's activity.

Similarly, the decrease-freq method does not blindly decrease the polling frequency. Rather, the frequency is gradually decreased over time until it returns to its original polling frequency. As with increase-freq, the Trend should consider a sensitivity factor, and more sensitive resources should be decreased more cautiously than the less sensitive.

Increase Frequency heuristics

When a resource is compromised, its polling frequency is immediately increased. Rationale: if a resource is actively being compromised, we will want to "keep an eye on it" in the short term to make sure it's not hit again quickly within the polling period.

Decrease Frequency heuristics

After a resource has been compromised and is "on the road to recovery," we will slowly decrease the polling frequency as follows:

Any change in frequency will occur NOT MORE THAN once per **original** period. That is, if the original period was 300s and polling frequency was tripled, the increase will be effective for at least the original period of 300s.

When an AgentThread is eligible for a slowdown (i.e., when the most recent change has been effective for at least the initial period), its history is consulted. If the Resource has a history of compromises, its polling frequency will NOT be decreased. The frequency is only decreased if the history does not indicate significant compromise.

Conclusions

The agent-based concepts developed and tested during this investigation provide timely, unattended restoration of the computing environment – whether it is a result of an information warfare attack or a system malfunction. The use of an autonomous resource monitor, coupled with rule-base logic to control the execution of recovery actions, provides maximum flexibility for this approach. The generic architecture that was designed and developed during this project provides the flexibility to adapt to the requirements and operational needs of *current* as well as *future* systems.

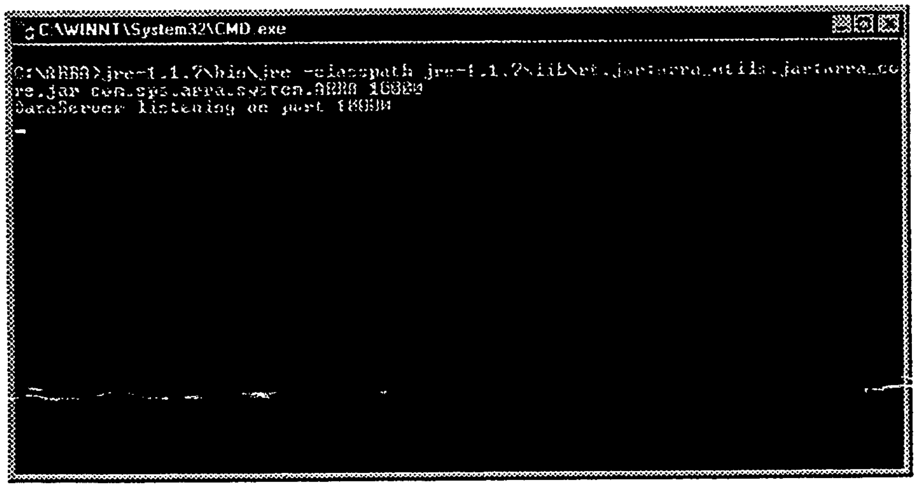
Testing provided evidence that most of the final implementation was transportable between diverse computing platforms and operating systems: Sun, SGI, PC – Unix, SunOS, Windows NT/95. There were some occasions, however, when the agent code did not perform as expected. These occasions usually involved low-level communication protocols or access to operating system features. The overall functioning of the generic agent components did work as expected on all platforms tested. File monitoring and recovery was implemented entirely using Java constructs, without any external or platform-specific code being developed. Process monitoring did require operating system-specific code to gather the status values needed to determine the state of application and operating system processes.

User's Manual

Automated Resource Recovery Agent (ARRA)

Starting the Agent

To start the agent in a Windows NT environment, open the ARRA folder that contains the ARRA agent and the Java Runtime Environment. Double-click the **ARRA.bat** file. This is a batch file that starts the execution of the agent. The window shown in figure 9 should be displayed.



```
C:\ARRA>jre-1.1.7\bin\jre -classpath jre-1.1.7\lib\rt.jar;arra_utils.jar;arra_co
re.jar com.sps.arra.system.ARRA 10000
DataServer listening on port 10000
_
```

Figure 9. Starting the Agent

The commands shown in the window start the Java RTE, load the agent classes, and listen on port 10000 for other agents on the local area network. Within a few seconds, the main ARRA agent window should appear, as shown in figure 10.

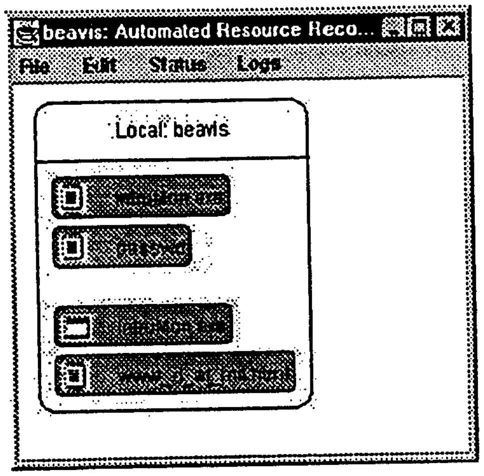


Figure 10. ARRA Agent Window

This window shows a view into an agent that has been pre-configured to monitor three files and one process. The three files are:

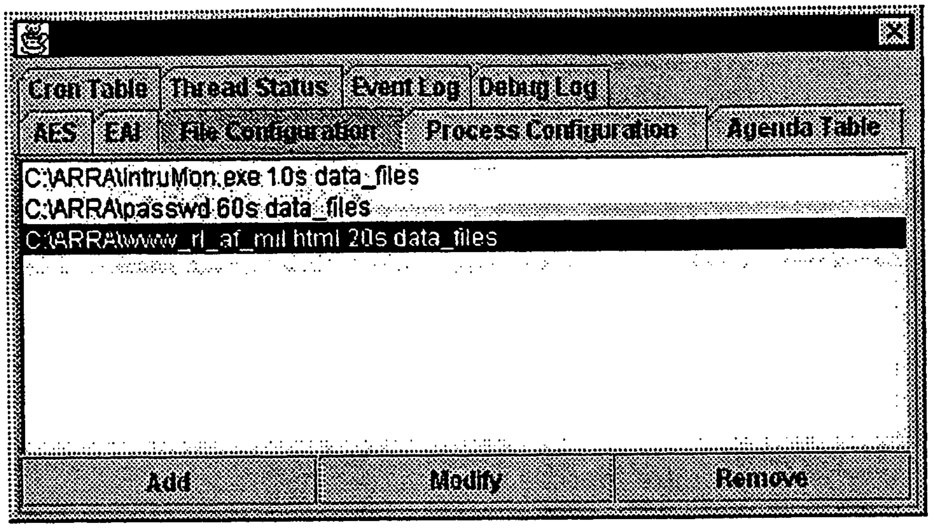
1) **intruMon.exe** - the binary for the intrusion monitoring process intruMon.exe

2) **passwd** – a password file

3) **www_rl_af_mil.html** – an HTML web page

Files are represented by the *document* icons commonly found in MS Windows environments. Similarly, *processes* are represented by the MS Windows *application* icons. Using these icons, we can distinguish between a binary executable file and the corresponding process.

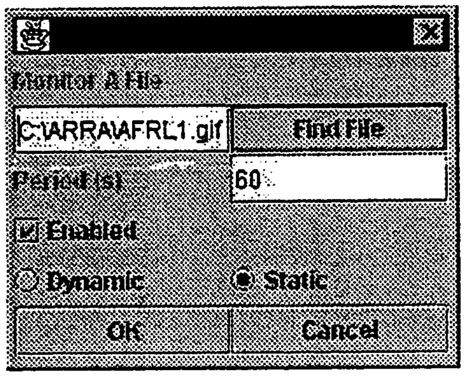
Several selections appear on the EDIT menu in the main window. Selecting *File Configuration* displays the configuration dialog shown in figure 11. This tabbed dialog is the interface for configuring the agent and for reviewing the actions taken by the agent during its execution. Selecting the *File Configuration* tab highlights the tab and displays the files that are currently being monitored by the agent. The buttons at the bottom of the window are used to add a new file to the agent for monitoring, modify an existing file being monitored, or remove the file from the agent's monitoring agenda.



```
C:\ARRA\IntruMon.exe 10s data_files
C:\ARRA\passwd 60s data_files
C:\ARRA\www_rl_af_mil html 20s data_files
```

Figure 11. File Configuration Dialog

Clicking the ADD button displays the window shown in figure 12.



Monitor A File

C:\ARRA\AFRL1.gif

Period (s) 60

[✓] Enabled

Dynamic [● Static]

Figure 12. Add Dialog

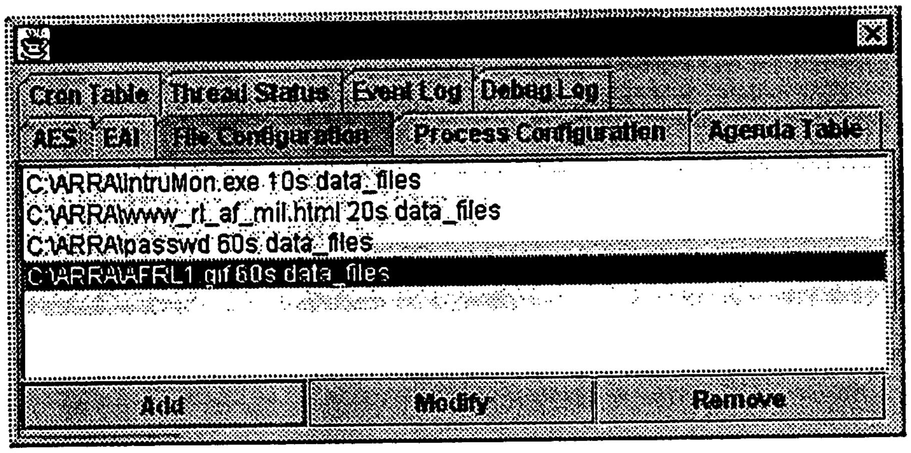
The name and path to the file can be entered in the text box in the upper left corner of the window or the FIND FILE button can be used to locate the new file to be monitored. The FIND FILE button will display a standard file dialog box where a file can be located using the point-and-click method.

If the ENABLED checkbox is selected (checked), the file will be monitored as soon as the OK button is clicked. If the ENABLED checkbox is not selected, the file will be added to the agent for *future* configurations – *without* immediate monitoring of the file. This feature is also useful if authorized modifications need to be made to a file. By deselecting the ENABLE checkbox, the file will be removed from the agent's agenda and the agent will not try to recover the file after modification.

The text box containing the number *60* indicates that the file will be monitored every 60 seconds. This field can be edited to specify the desired monitoring frequency.

The two radio buttons labeled DYNAMIC and STATIC indicate to the agent the *type* of file that is being added to the agenda: *dynamic* – means the file can be modified by an authorized process, or *static* – means the file should never change while being monitored. Dynamic files require special cooperation between the process that is dynamically modifying the file and the agent.

The window displayed in figure 13 indicates that the file C:\ARRA\ADRL1.GIF has been added to the agent's agenda and is being monitored every 60 seconds.

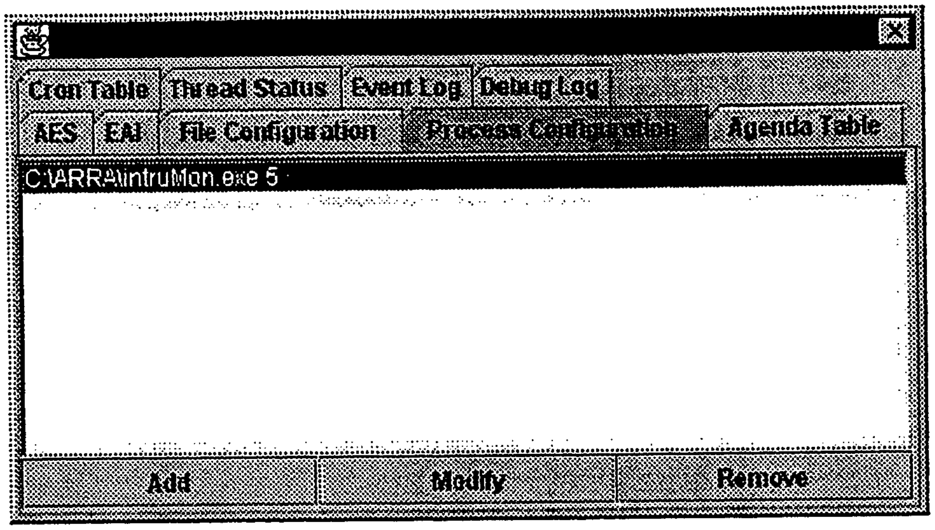


```
C:\ARRA\IntruMon.exe 10s data_files
C:\ARRA\www_rl_af_mil.html 20s data_files
C:\ARRA\passwd 60s data_files
C:\ARRA\AFRL1.gif 60s data_files
```

Figure 13. Updated File Configuration Dialog

The MODIFY button allows the currently highlighted file entry to be modified using a series of dialogs similar to those shown for the ADD button. The REMOVE button allows the currently highlighted entry to be removed from the agent's agenda.

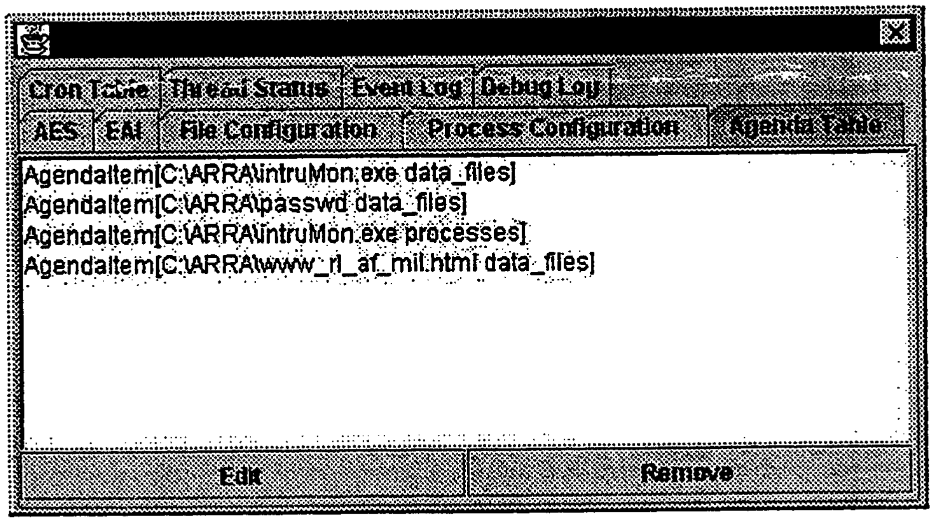
Selecting the *Process Configuration* tab in the Configuration Dialog or selecting the EDIT menu in the main window enables you to update the *processes* in the agent's agenda. This sequence is similar to the File Configuration sequence we just discussed. It displays a Configuration Dialog with ADD, MODIFY, and REMOVE buttons that have same effect, except they apply to *processes* instead of *files.* A file selection dialog appears for the selection of the binary executable to monitor. If the process is not active in the system, the agent starts the process. Adding a process to the agent's agenda causes both the process *and* the binary executable to be added. The static binary file will appear in the File Configuration dialog. The Process Configuration window in figure 14 shows that the **intruMon.exe** process is being monitored every five seconds.



C:\ARRA\intruMon.exe 5

Figure 14. Process Configuration Dialog

Selecting the Agenda Table tab will cause a dialog similar to the following to appear. This window shows everything that is on the agent's agenda and contains entries from both the file and process configuration windows.



```
AgendaItem[C:\ARRA\intruMon.exe data_files]
AgendaItem[C:\ARRA\passwd data_files]
AgendaItem[C:\ARRA\intruMon.exe processes]
AgendaItem[C:\ARRA\www_rl_af_mil.html data_files]
```

Figure 15. Updated Process Configuration Dialog

When a file or process compromise is detected by the agent, several things happen. First, the agent begins to execute the necessary actions to recover the compromised resource. Next, the icon in the main window that represents the compromised resource changes color to red – indicating that the agent has detected a problem. For example, the window in figure 16 shows a red icon for the HTML file.

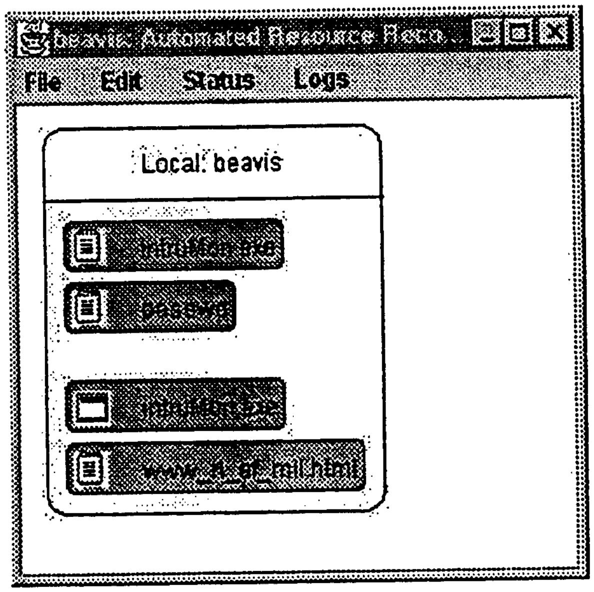
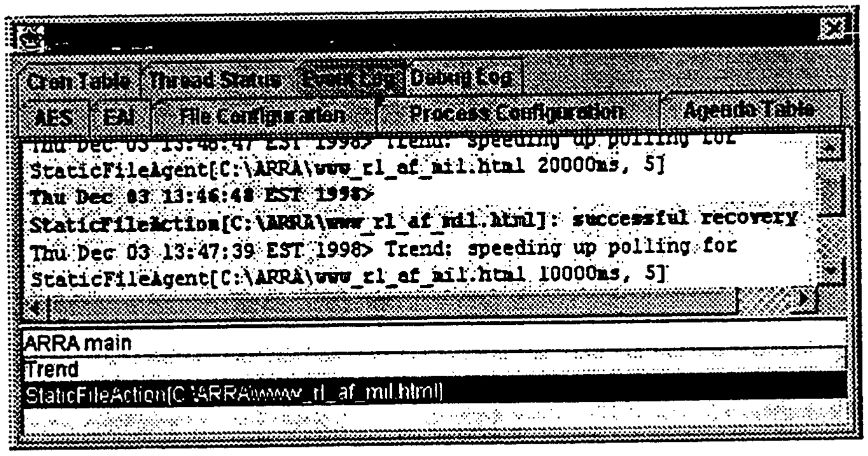


Figure 16. Visual Alert of an Agent-Detected Problem

The recovery action taken by the agent is written to the Event Log as shown below.

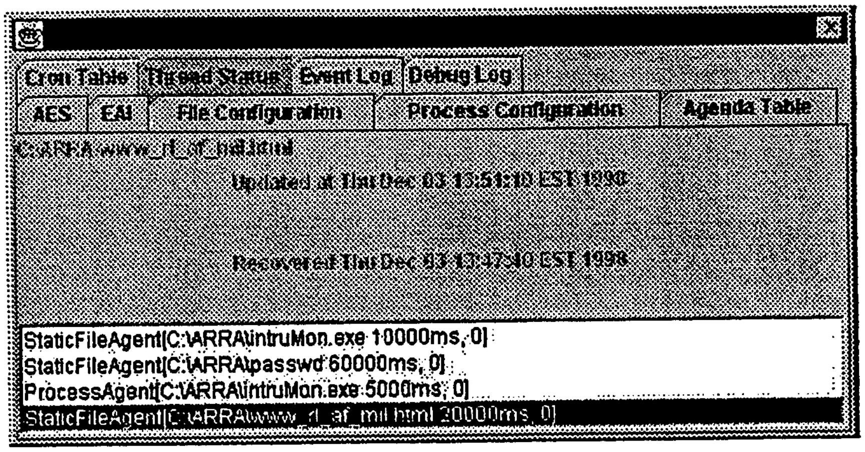


```
Thu Dec 03 13:46:47 EST 1998> Trend: speeding up polling for
StaticFileAgent[C:\ARRA\www_rl_af_mil.html 20000ms, 5]
Thu Dec 03 13:46:48 EST 1998>
StaticFileAction[C:\ARRA\www_rl_af_mil.html]: successful recovery
Thu Dec 03 13:47:39 EST 1998> Trend: speeding up polling for
StaticFileAgent[C:\ARRA\www_rl_af_mil.html 10000ms, 5]
```

```
ARRA main
Trend
StaticFileAction[C:\ARRA\www_rl_af_mil.html]
```

Figure 17. Event Log

The status of the compromised resource can be viewed by first selecting the Thread Status tab and then clicking on the resource entry of interest.



```
C:\ARRA\www_rl_af_mil.html
              Updated at Thu Dec 03 13:51:10 EST 1998

              Recovered Thu Dec 03 13:47:40 EST 1998

StaticFileAgent[C:\ARRA\intruMon.exe 10000ms, 0]
StaticFileAgent[C:\ARRA\passwd 60000ms, 0]
ProcessAgent[C:\ARRA\intruMon.exe 5000ms, 0]
StaticFileAgent[C:\ARRA\www_rl_af_mil.html 30000ms, 0]
```

Figure 18. Compromised Resource Status

Finally, a message is broadcast to the other agents on the local area network to inform them of the compromise.

Stopping the Agent

Each of the tabs in the dialog have identical counterparts in the main window with the exception of the EXIT menu item. To terminate execution of the agent, select the EXIT menu item under the File menu. This will cause the agent to stop execution and all the windows including the batch file window will be removed from the display. The agent saves its current configuration before terminating and will reestablish the configuration and populate the agenda upon the next execution.

Time Dependent Monitoring: CRON

Modeled after the UNIX "cron" facility, the ARRA includes a "CronEntry" object that provides an added dimension to the standard AgendaItem.

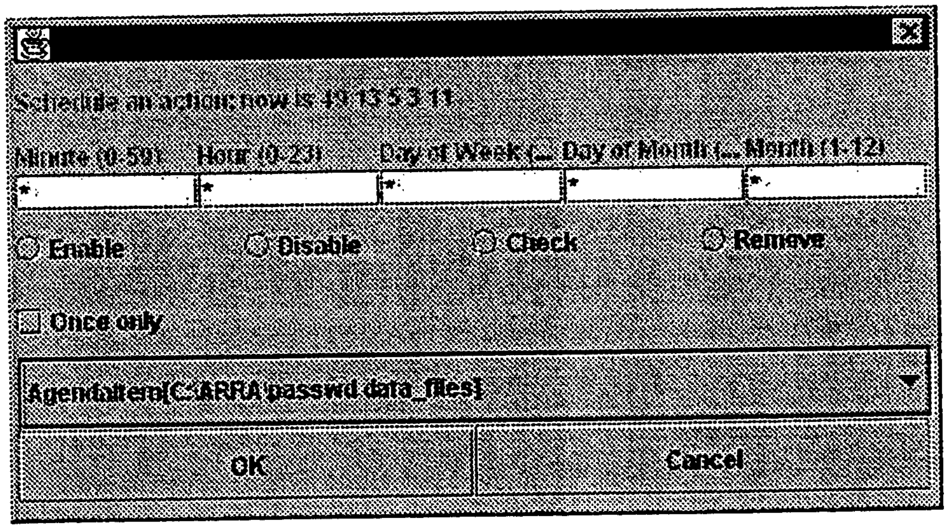
The UNIX cron facility consists of several components – the "crontab," a table of times and commands, and the "crond," a "cron daemon" that runs constantly, executing commands as they come due. A "time" in cron's language is a quintuple representing the minute (0-59), hour (0-23), day of week (Sunday –Saturday, 0-6), day of month (1-31), and month of year (1-12). Perhaps they should consult the Mac guys, who had it solved a long time ago!!) Any field may contain single integers, a range of integers (first-last, e. g. 1-5), a wildcard '*', or a comma-separated list of the three, in any combination. Valid entries in the "day of week" field could be "*" (any day), "1-5" (Monday-Friday), "1,3,5" (Monday, Wednesday, Friday), "0,1,3-6" or "0-1,3-6" or "0,1,3,4,5,6" (any day except Tuesday). An inversion operator, such as "!2" (not Tuesday) is not provided. Clearly, the UNIX cron facility's time-specification mechanism is extremely powerful.

Note: the UNIX crond uses the "day of week" field in an interesting manner. The "day of week" and "day of month" do not coincide – they operate independently, and the rule will fire if **either** is true. That is, a "1" day of week and "1" day of month will fire both every Monday **and** on the first of every month (but only once if the first is a Monday). Further, with UNIX there is no "good" way to have a rule that fires on the first Monday of every month. The accepted UNIX method uses a Monday rule and a script that will check to see if today's date is less than 8 – presumably, Monday will only occur once in the first 7 days of the month. ARRA's scheme is slightly different, making the two "day" fields dependent on one another, should allow one to have a rule that fires on "the second Tuesday." A rule like "the first Tuesday after the first Monday" would work, too – "* * 2 2-8 *".

Each entry in the cron table consists of the time specifiers and a command to be executed. UNIX cron executes commands using the Bourne Shell, a robust command language that gives access to virtually any UNIX commands. The UNIX cron facility does not specify the order of execution of entries in the cron table.

ARRA's modeling of the UNIX cron facility uses a single CronD object (which models the crond daemon) and some collection of CronEntry objects (a five-component time specifier, a "once only" flag, an action to be executed, and an AgendaItem on which to operate).

The CronEntry has four major parameters: a five-component time field, a command, a "once-only" flag, and an AgendaItem as shown in figure 19.



Schedule an action; now is 49 13 5 3 11

Minute (0-59)	Hour (0-23)	Day of Week (...	Day of Month (...	Month (1-12)
*	*	*	*	*

AgendaItem[C:\ARRA\passwd data_files]

Figure 19. CronEntry Parameters Dialog

The command can be ENABLE, DISABLE, CHECK, or REMOVE. The CronEntry, when instructed to execute, will perform the specified command on its AgendaItem.

1. ENABLE: Enable the AgendaItem
2. DISABLE: Disable the AgendaItem
3. CHECK: Perform the appropriate analysis for the AgendaItem (**note**: this calls the analyze() method in the AgentThread corresponding to the AgendaItem; as specified, the analyze() method will not proceed if the AgendaItem is disabled. Therefore, an attempt to CHECK a disabled AgendaItem will fail silently)
4. REMOVE: Remove the AgendaItem from the Agenda (**note**: this does not remove the CronEntry from the CronTable; subsequent attempts to REMOVE can fail because the specific AgendaItem may not exist, or may have been removed and then re-added, causing this CronEntry to "lose track" of it. This special case should be examined more closely, and precautions should be taken)

As with the UNIX cron, the order of execution of cron entries is undefined. That means that three entries to successively ENABLE, CHECK, and DISABLE an AgendaItem will probably fail.

The "one shot" flag is used to emulate the functionality of the UNIX "at" command. "at" submits a normal job to the UNIX cron facility, but the job is only executed once, then removed from the cron table. Similarly, and CronEntry flagged for use "once only" removes itself from the CronD upon execution.

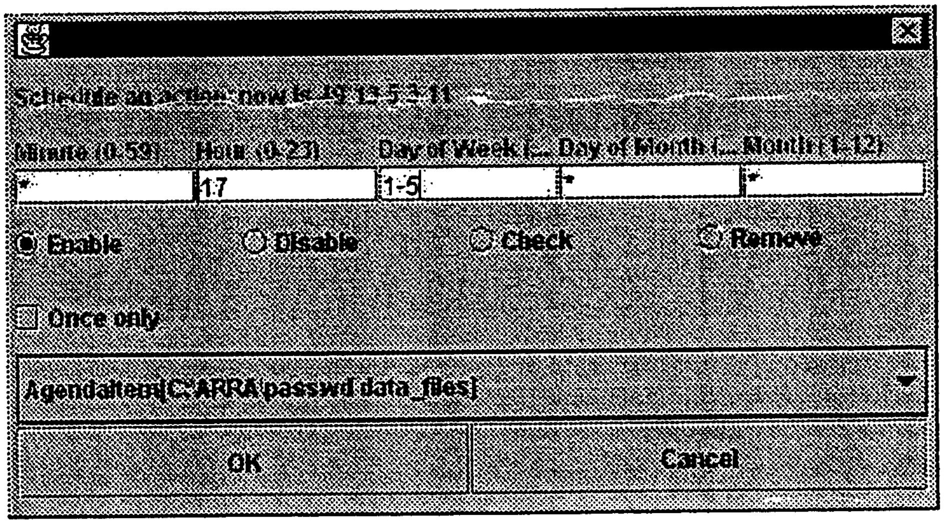
Examples

For the following examples, only the time and action fields are given. "False" is assumed for the "once only" flag, and the appropriate AgendaItem is assumed.

To monitor some resource after normal business hours:

> *Strategy: given an AgendaItem that will periodically monitor a resource, we want to enable it every weekday evening (hour 17, 5pm), and disable it during working hours (hour 7,7am). The time specifiers for the rules are as follows:*
>
> - 0 17 1-5 * * ENABLE
> - 0 7 1-5 * * DISABLE
>
> *Alternate strategy: have an AgendaItem that will poll at some fixed frequency constantly, but on the quarter hour after business hours:*
>
> - 0,15,30,45 17-7 1-5 * * CHECK

An example of a timed enable of the password file is illustrated in figure 20.



Schedule an action: now is 49 13 5 3 11

Minute (0-59)	Hour (0-23)	Day of Week (...	Day of Month (...	Month (1-12)
*	17	1-5	*	*

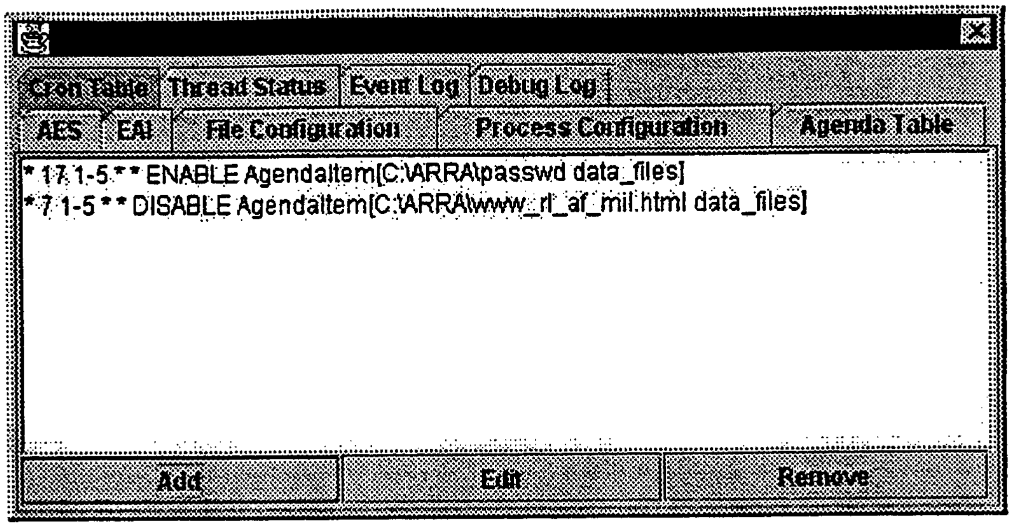
[✓ Enable] Disable Check Remove

☐ Once only

AgendaItem[C:\ARRA\passwd data_files]

Figure 20. Timed Enable Example

Selecting the OK button returns the display to the tabbed dialog where the current timed configuration is shown. The dialog shows both an ENABLE and a DISABLE command, as previously discussed.



```
* 17 1-5 * * ENABLE AgendaItem[C:\ARRA\passwd data_files]
* 7 1-5 * * DISABLE AgendaItem[C:\ARRA\www_rl_af_mil.html data_files]
```

Figure 21. Dialog Showing Enable & Disable Commands

As in the File and Process configurations, entries can be added, edited, and removed from the agent's agenda.

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and technology for aerospace command and control and its transition to air, space, and ground systems to meet customer needs in the areas of Global Awareness, Dynamic Planning and Execution, and Global Information Exchange is the focus of this AFRL organization. The directorate's areas of investigation include a broad spectrum of information and fusion, communication, collaborative environment and modeling and simulation, defensive information warfare, and intelligent information systems technologies.